Terry Velasquez

The Database:
I used the CREATE DATABASE hospital_portal to create the database and then the USE hospital_database to make sure it was the database I was using at the time. Next, we will use the CREATE TABLE command to create three tables. In order to make patient_id the PRIMARY KEY and set it to AUTO_INCREASE, which is how we will use all primary key ids in the rest of the database, the patients table is created with patient_id, patient_name, age, admission_date, and discharge_date as its attributes. Here is the doctors table, where the primary key is doctor_id and the attributes are doctor_name, doctor_field, and doctor_id. Lastly, we create the appointments table with the following fields: patient, doctor, appointment_date, appointment_time, and appointment_id (PRIMARY KEY). However, making sure that patient_id and doctor_id are referenced to the appropriate tables and set as FOREIGN KEYs this time. Here are some screenshots showing the creation and initial populating of these tables before using the server to manage that going forward.

```
1  CREATE DATABASE hospital_portal;
2  USE hospital_portal;
3
4  CREATE TABLE patients (
5      patient_id int not null unique auto_increment primary key,
6      patient_name varchar(45) NOT NULL,
7      age int not null,
8      admission_date date,
9      discharge_date date
10     );
11
12
13 CREATE TABLE Appointments (
14     appointment_id int not null unique auto_increment primary key,
15     patient_id int not null,
16     doctor_id int not null,
17     appointment_date date not null,
18     appointment_time decimal not null,
19     foreign key (patient_id) references patients(patient_id)
20     );
21
22 INSERT INTO patients (patient_name, age, admission_date, discharge_date)
23 VALUES ("Maria Jozef", 67, "2023/10/01", "2023/10/07"),
24        ("Breanna Olsen", 35, "2023/06/25", "2023/06/29"),
25        ("Henry Smith", 55, "2023/09/17", "2023/09/25");
26
27 SELECT * FROM patients;
28
29 CREATE TABLE doctors (
30     doctor_id int not null unique auto_increment primary key,
31     doctor_name varchar(45) NOT NULL,
32     doctor_field varchar(45) NOT NULL
33     );
34
```

```
34
35 •    ALTER TABLE appointments
36      ADD FOREIGN KEY (doctor_id) REFERENCES doctors(doctor_id);
37
```

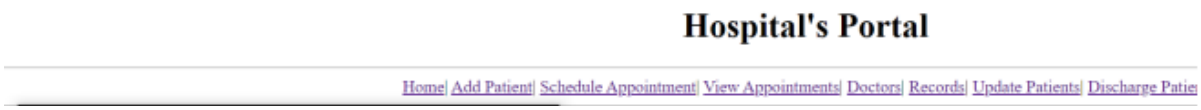
```
69
70 •   INSERT INTO doctors (doctor_name, doctor_field)
71      VALUES ("Steven Strange", "Neural Surgeon"),
72             ("Wanda Maximoff", "Pediatrics"),
73             ("Vladimir Tepes", "Cardiology");
```

The next step in the database process is to create two stored procedures, scheduleAppointments, dischargePatient, and updatePatient, to help with specific actions. First, we create a ScheduleAppointment using a DELIMITER and the CREATE PROCEDURE command. This appointment type takes four parameters: app_patient_id, app_doctor_id, app_appointment_date, and app_appointment_time. Putting "app" in front of each parameter just helps to distinguish it from the attribute into which it will be inserted. Next, we use INSERT INTO to insert the value of the parameter into each corresponding attribute within the appointments table. With only one parameter (app_patient_id) to tell the UPDATE call WHERE it should be making the changes and SET the date to today, the DischargePatient procedure is much shorter and easier to understand than the previous one. Arrived us with the call "UPDATE patients SET patient_id = app_patient_id; WHERE discharge_date = CURRENT_DATE()." At last, I created an UpdatePatient procedure, which required five parameters that matched the set of attributes found in a patient entry. However, this time, the patient to be updated was identified by the patient_id, and the remaining parameters were the values that needed to be changed.

```
160     DELIMITER //
161
162 •   CREATE PROCEDURE UpdatePatient(
163         IN p_patient_id INT,
164         IN p_patient_name VARCHAR(45),
165         IN p_age INT,
166         IN p_admission_date DATE,
167         IN p_discharge_date DATE
168     )
169     BEGIN
170         UPDATE patients
171         SET
172             patient_name = p_patient_name,
173             age = p_age,
174             admission_date = p_admission_date,
175             discharge_date = p_discharge_date
176         WHERE patient_id = p_patient_id;
177     END //
178
179     DELIMITER ;
180
```

```
54
55
56      DELIMITER //
57
58  •   CREATE PROCEDURE DischargePatient (
59          IN app_patient_id INT
60          )
61      BEGIN
62          UPDATE patients
63          SET discharge_date = CURRENT_DATE()
64          WHERE patient_id = app_patient_id;
65
66      END //
67
68      DELIMITER ;
---
160     DELIMITER //
161
162 •   CREATE PROCEDURE UpdatePatient(
163         IN p_patient_id INT,
164         IN p_patient_name VARCHAR(45),
165         IN p_age INT,
166         IN p_admission_date DATE,
167         IN p_discharge_date DATE
168     )
169     BEGIN
170         UPDATE patients
171         SET
172             patient_name = p_patient_name,
173             age = p_age,
174             admission_date = p_admission_date,
175             discharge_date = p_discharge_date
176         WHERE patient_id = p_patient_id;
177     END //
178
179     DELIMITER ;
```

Database to Server Connection: In order to link the database to our Python code, I must first download and install mysql.connector. I then manage to establish a successful connection by changing the password from portalDatabase.py to my database password, enabling the portal to operate on my browser and retrieve data from the database.



The portalDatabase.py server and methods I chose to utilize the addPatient and getAllPateints methods as my model for the remaining methods because they were largely finished for us. Using the code from addPatient and getAllPatient as a guide, I then implemented the methods scheduleAppointment(), viewAppointments(), dischargePatient(), viewAllDoctors(), viewRecords(), and updatePatient(), making sure to modify the query to the appropriate queries only. Every code is displayed below:

```
def scheduleAppointment(self, patient_id, doctor_id, appointment_date, appointment_time):

    # Implement the functionality
    if self.connection.is_connected():
        self.cursor = self.connection.cursor()
        query = "CALL ScheduleAppointment(%s, %s, %s, %s);"
        self.cursor.execute(query, (patient_id, doctor_id, appointment_date, appointment_time))
        self.connection.commit()
        return

def viewAppointments(self):

    # Implement the functionality

    if self.connection.is_connected():
        self.cursor = self.connection.cursor()
        query = "SELECT * FROM appointments"
        self.cursor.execute(query)
        records = self.cursor.fetchall()
        return records


def dischargePatient(self, patient_id):

    # Implement the functionality
    if self.connection.is_connected():
        self.cursor = self.connection.cursor()
        query = "CALL DischargePatient(%s);"
        self.cursor.execute(query, (patient_id))
        self.connection.commit()
        return
# Add more methods as needed for hospital operations
```

```
        return
# Add more methods as needed for hospital operations

def viewAllDoctors(self):
    if self.connection.is_connected():
        self.cursor = self.connection.cursor()
        query = "SELECT * FROM doctors"
        self.cursor.execute(query)
        records = self.cursor.fetchall()
        return records

def viewRecords(self):
    if self.connection.is_connected():
        self.cursor = self.connection.cursor()
        query = "SELECT * FROM recordsview"
        self.cursor.execute(query)
        records = self.cursor.fetchall()
        return records

def updatePatient(self, patient_id, patient_name, age, admission_date, discharge_date,):
    if self.connection.is_connected():
        self.cursor = self.connection.cursor()
        query = "CALL UpdatePatient(%s, %s, %s, %s, %s);"
        self.cursor.execute(query, (patient_id, patient_name, age, admission_date, discharge_date))
        self.connection.commit()
        return
```

I had to finish developing the webpages for the do_GET section of the server where the user would be redirected when they clicked on any of the hyperlinks in the header, since at first they were pointing to blank pages. Given that Home has already displayed a list of every patient. I took that code and modified it to use the viewAppointment() method instead, which caused the appointments table to pull data from the database and display appointments rather than patients in a very similar list. The corresponding actions were taken for viewAllDoctors and viewRecords.

```
#View Doctors code
if self.path == '/viewAllDoctors':
    data=[]
    records = self.database.viewAllDoctors()
    print(records)
    data=records
    self.send_response(200)
    self.send_header('Content-type','text/html')
    self.end_headers()
    self.wfile.write(b"<html><head><title> Hospital's Portal </title></head>")
    self.wfile.write(b"<body>")
    self.wfile.write(b"<center><h1>Hospital's Portal</h1>")
    self.wfile.write(b"<hr>")
    self.wfile.write(b"<div> <a href='/'>Home</a>| \
                    <a href='/addPatient'>Add Patient</a>|\
                     <a href='/scheduleAppointment'>Schedule Appointment</a>|\
                     <a href='/viewAppointments'>View Appointments</a>|\
                     <a href='/viewAllDoctors'>Doctors</a>|\
                     <a href='/viewRecords'>Records</a>|\
                     <a href='/updatePatient'>Update Patients</a>|\
                     <a href='/dischargePatient'>Discharge Patient</a></div>")
    self.wfile.write(b"<hr><h2>All Doctors</h2>")


    self.wfile.write(b"<table border=2> \
                        <tr><th> Doctor ID </th>\
                            <th> Doctor Name </th>\
                            <th> Medical Field </th</tr>")

    for row in data:
        self.wfile.write(b' <tr> <td>')
        self.wfile.write(str(row[0]).encode())
        self.wfile.write(b'</td><td>')
        self.wfile.write(str(row[1]).encode())
        self.wfile.write(b'</td><td>')
        self.wfile.write(str(row[2]).encode())
        self.wfile.write(b'</td></tr>')

    self.wfile.write(b"</center></body></html>")
    return
```

In order to make it easier to look at the list of patients when deciding which patient needs to be discharged or updated, I decided to include a list of all the patients along with the forms for the dischargePatient() and updatePatient() functions. After all, those methods only require an ID when selecting a patient, so having the list of patients' IDs and the rest of their information on one page would be very helpful. Here's an illustration of that code:

```
if self.path == '/updatePatient':
    data=[]
    records = self.database.getAllPatients()
    print(records)
    data=records
    self.send_response(200)
    self.send_header('Content-type','text/html')
    self.end_headers()
    self.wfile.write(b"<html><head><title> Hospital's Portal </title></head>")
    self.wfile.write(b"<body>")
    self.wfile.write(b"<center><h1>Hospital's Portal</h1>")
    self.wfile.write(b"<hr>")
    self.wfile.write(b"<div> <a href='/'>Home</a>| \
                     <a href='/addPatient'>Add Patient</a>|\
                      <a href='/scheduleAppointment'>Schedule Appointment</a>|\
                      <a href='/viewAppointments'>View Appointments</a>|\
                      <a href='/viewAllDoctors'>Doctors</a>|\
                      <a href='/viewRecords'>Records</a>|\
                      <a href='/updatePatient'>Update Patients</a>|\
                      <a href='/dischargePatient'>Discharge Patient</a></div>")

    self.wfile.write(b"<table border=2> \
                        <tr><th> Patient ID </th>\
                            <th> Patient Name</th>\
                            <th> Age </th>\
                            <th> Admission Date </th>\
                            <th> Discharge Date </th></tr>")
    for row in data:
        self.wfile.write(b' <tr> <td>')
        self.wfile.write(str(row[0]).encode())
        self.wfile.write(b'</td><td>')
        self.wfile.write(str(row[1]).encode())
        self.wfile.write(b'</td><td>')
        self.wfile.write(str(row[2]).encode())
        self.wfile.write(b'</td><td>')
        self.wfile.write(str(row[3]).encode())
        self.wfile.write(b'</td><td>')
        self.wfile.write(str(row[4]).encode())
        self.wfile.write(b'</td></tr>')


    self.wfile.write(b"<hr><h2>Update Patient</h2>")


    self.wfile.write(b"<form action='/updatePatient' method='post'>")
    self.wfile.write(b'<label for="patient_id">ID of patient to be updated:</label>\
          <input type="number" id="patient_id" name="patient_id" required><br><br>\
          <label for="patient_name">Patient Name:</label>\
          <input type="text" id="patient_name" name="patient_name"><br><br>\
          <label for="patient_age">Age:</label>\
          <input type="number" id="patient_age" name="patient_age"><br><br>\
          <label for="admission_date">Admission Date:</label>\
          <input type="date" id="admission_date" name="admission_date"><br><br>\
          <label for="discharge_date">Discharge Date:</label>\
          <input type="date" id="discharge_date" name="discharge_date"><br><br>\
          <input type="submit" value="Submit">\
          </form>')
```

Finally, I applied the same method I employed for the portalDatabase code to the portalSever.py file. I was able to create template code for the remaining actions under do_POST by using the addPatient code that was already provided to us under do_POST and the code for the home screen that displayed a list of all the patients. As an illustration, you can see that I created a post screen using the addPatient template whenever a patient's information is updated, an appointment is made, or the patient is discharged. This confirms that the actions were successful and gives the patient the choice to repeat the action or go to a different area of the portal.

```
#Update Patient code start

try:
    if self.path == '/updatePatient':
        self.send_response(200)
        self.send_header('Content-type','text/html')
        self.end_headers()
        form = cgi.FieldStorage(
            fp=self.rfile,
            headers=self.headers,
            environ={'REQUEST_METHOD': 'POST'}
        )

        patient_id = int(form.getvalue("patient_id"))
        patient_name = form.getvalue("patient_name")
        age = int(form.getvalue("patient_age"))
        admission_date = form.getvalue("admission_date")
        discharge_date = form.getvalue("discharge_date")

        self.database.updatePatient(patient_id, patient_name, age, admission_date, discharge_date)

        self.wfile.write(b"<html><head><title> Hospital's Portal </title></head>")
        self.wfile.write(b"<body>")
        self.wfile.write(b"<center><h1>Hospital's Portal</h1>")
        self.wfile.write(b"<hr>")
        self.wfile.write(b"<div> <a href='/'>Home</a>| \
                         <a href='/addPatient'>Add Patient</a>|\
                          <a href='/scheduleAppointment'>Schedule Appointment</a>|\
                          <a href='/viewAppointments'>View Appointments</a>|\
                          <a href='/viewAllDoctors'>Doctors</a>|\
                          <a href='/viewRecords'>Records</a>|\
                          <a href='/updatePatient'>Update Patients</a>|\
                          <a href='/dischargePatient'>Discharge Patient</a></div>")
        self.wfile.write(b"<hr>")
        self.wfile.write(b"<h3>Patient information updated</h3>")
        self.wfile.write(b"<div><a href='/updatePatient'>Update Another Patient</a></div>")
        self.wfile.write(b"</center></body></html>")

except IOError:
    self.send_error(404, 'File Not Found: %s' % self.path)
```